Язык программирования Java

Лекция 1

Перевод курса CS11 Java Track

Краткая история языка Java

- Создан компанией Sun Microsystems в конце 90-х годов
  - Предназначался для программирования встраиваемых систем
  - Основная задача разработки - сделать язык лучше чем C++
  - Переименован в Java в 1994
- Версия Java 1.0 выпущена в 1995 году
  - Версии 1.1, 1.2, 1.3. 1.4
- Схема нумерации версий изменена в Java 5.0
  - (версия для разработчиков и SDK по прежнему имеет номер 1.5)
- Текущая рабочая версия Java 7
  - Java 8 сейчас выпущена в версии для разработчиков, для ознакомления

Краткая история языка Java (2)

- Язык и стандартные библиотеки сильно развились с годами
  - В версии Java 6 выпущенной в конце 2006 г. появилось много новых свойств языка и новые API функции
  - Java 7 выпущена в середине 2011 года
- Java (в основном) переведена в свободно распространяемый код компанией Sun в 2007 году
  - Это позволило перенести и адаптировать Java под разные аппаратные платформы
- В январе 2010 г. компания Oracle поглотила Sun
  - Это вызвало значительное беспокойство о будущем Java

Цель разработки языка Java

- Java имеет простой и знакомый синтаксис
  - Основан на синтаксисе языка C++, но без многих тонкостей, имеющихся в языке C++
- Объектно-ориентированный язык
  - Хорошо подходит для разработки распределенных систем.
- Платформо-независимый язык
  - Исходный код и исполняемые файлы легко переносятся на разные программно-аппаратные платформы
- Динамическая загрузка и связывание
  - Уменьшает количество рекомпиляций кода и облегчает создание модульных приложений
- Безопасность использования кода
  - Поддерживается технологиями проверки классов, цифровой подписи кода, наборами разрешений
- Многозадачность
  - Язык обеспечивает платформо-независимую поддержку исполнения потоков

Как работает Java

- Исходный код хранится в файлах .java
- В одном файле – один класс
  - Класс может содержать вложенные классы, но “корневой” класс в файле только один
- Имя этого класса определяет имя файла, например:

```
HelloWorldApp.java
// Выводим на консоль сообщение и завершаем работу.
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

Как работает Java (2)

- Компилятор Java компилирует исходный код .java в платформо-независимый исполняемый код и помещает его в файлы с расширением .class.
  - Команда javac HelloWorldApp.java

    создает файл HelloWorldApp.java
  - Эти файлы содержат так называемый байт-код – инструкции для виртуальной машины Java (Java Virtual Machine / JVM).
- В нашем примере байт-код будет такой:

```
public static void main(java.lang.String[])
0: getstatic #2;      //Поле java/lang/System.out
3: ldc #3;               //Строка "Hello, world!"
5: invokevirtual #4;  //Метод java/io/PrintStream.println
8: return
```

Как работает Java (3)

- Программа запускается с помощью виртуальной машины Java (JVM)
  - В командной строке указать имя класса, вместо имени файла программы:

```
> java HelloWorldApp
Hello, world!
```

- Программа java запускает виртуальную машину JVM
  - Java можно запустить на многих платформах, для которых сделаны JVM
    - Windows, Linux, Solaris, Mac OS X
- Некоторые JMV для повышения производительности компилируют байт-код в исполняемый код платформы
  - Это называется компиляция во время исполнения (Just-in-time/JIT)

Комментарии

- Правила использования комментариев Java такие же как в языке C

```
/*
 * Этот метод печатает Hello, world!
 */
public static void main(String[] args) {
  // Эта строчка довольно запутанная ...
  System.out.println("Hello, world!"); // готово!
}
```

  - Блочные комментарии могут занимать несколько строк
  - Однострочные комментарии заканчиваются вместе с концом строки
- Вложенные /* */ комментарии не допускаются
  - Блочный комментарий заканчивается с первым встреченным */

Типы данных Java

- Примитивные типы:

boolean	значение true или false
char	16-ти битовое целое без знака (для символов Unicode)
byte	8-ми битовое целое со знаком
int	32-х битовое целое со знаком
long	64-х битовое целое со знаком
float	32-х битовое значение с плавающей точкой
double	64-х битовое значение с плавающей точкой

Другие типы данных

- Типы указателей (ссылки/reference) на объекты
  - На примитивные типы эти ссылки указывать не могут
  - Значение переменной ссылочного типа может быть равно null если она ни на что не указывает
  - Примеры: String и Integer
- Массивы Java тоже ссылочные типы

```
int[] numArray; // лучше объявлять массив так!
int numArray[]; // тоже работает
```

  - Подробнее познакомимся с массивами в следующей лекции

Литералы

- Логические константы это true и false
- Целые значения указываются так:

```
int i = 17;
```

- Значения типа long имеют суффикс L:

```
long secondsInYear = 31556926L;
```

  - Не следует использовать для обозначения константы букву ” l” в нижнем регистре, так как она похожа на единицу.

- По умолчанию константа с плавающей точкой имеет тип double:

```
double pi = 3.14159265358979323;
```

- Для констант типа float указывается суффикс F:

```
float goldenRatio = 1.618f;
```

  - В данном случае можно использовать суффикс F или f

Символы и строки

- Символьные константы задаются литерами в одинарных кавычках или числами:

```
char capA = 'A'; // предпочтительно
char capA = 65; // как правило, менее удобно для понимания текста
```

- Строковые константы заключаются в двойные кавычки:

```
String sandwichType = "pastrami";
```

- Служебные символы должны вставляться с помощью escape последовательности:

```
String msg = "Он сказал, \"Java это здорово!\"";
```

  - В строках часто встречаются служебные символы:

```
\t = табуляция \r = возврат каретки \n = перевод строки
\\ = обратная косая черта \' = одинарная кавычка
\" = двойная   кавычка
```

Правила назначения имен в Java

- Имена должны начинаться с буквы и содержать только буквы и цифры
  - _ и $ считаются в Java “буквами”
  - Не используйте символ $ в именах, потому что компилятор его использует для автогенерации кода.
- Регистр очень важный элемент стиля написания программ Java
  - Имена полей и методов должны соответствовать соглашению camelCase. То есть имя должно начинаться с маленькой буквы и, если оно составлено из нескольких слов, то каждое последующее слово должно начинаться с большой буквы.
  - Имена классов и интерфейсов должны соответствовать соглашению UpperCamelCase.
  - Имена модулей всегда следует указывать в нижнем регистре.
- Язык Java имеет ряд общепринятых в промышленности правил и соглашений, касающихся стиля написания программ
  - Их необходимо изучить и следовать им
  - Им необходимо строго следовать, выполняя задания этого курса.

Переменные и начальные значения

- Переменные в Java объявляются также как в C/C++

```
int i;
boolean err = false, done;
String name = "Donnie";
```

- Локальные переменные не инициализируются автоматически

```
int i;
i = i + 1;
```

  - Такой код: будет компилироваться с ошибкой:

```
“variable i might not have been initialized / переменная i не инициализирована”
```

    - Это пример того как тщательно Java следит за корректностью кода
    - Такой код написанный на C/C++ компилируется без ошибок

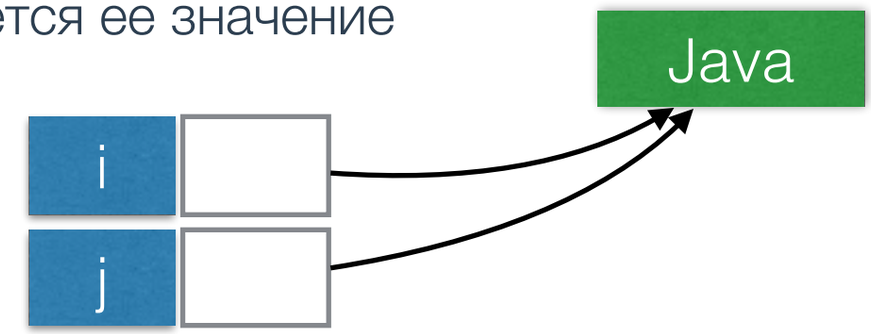
Переменные примитивных и ссылочных типов

- Разница между переменными примитивных и ссылочных типов заключается в способе хранения их значений
- Примитивные типы

```
int i = 20;
int j = i;
```

  - В переменной примитивного типа охраняется ее значение
- Ссылочные типы

```
String s1 = "Java!";
String s2 = s1;
```



  - В переменной ссылочного типа хранится ссылка (указатель) на значение, хранящееся в основной памяти.
  - Разные переменные ссылочного типа могут указывать на одно и то же значение.

Операторы Java

- Java имеет тот же набор операторов что C/C++:
  - Простые арифметические операторы: + - * / %
  - Составные присвоения: += -= *= /= и т.п.
  - Инкремент/декремент: -- ++ (пред и пост)

```
int i = 5;
int j = ++i; // j = 6, i = 6
int k = i++; // k = 6, i = 7
```

- Сравнения: == != < <= > >=
  - Операторы сравнения возвращают значения логического типа (boolean)
  - Значения других типов нельзя преобразовать в логический тип (как, например, часто делают с типом int в выражениях C/C++)
  - И наоборот, тип boolean в Java нельзя преобразовать в другой тип

Логические операторы

- Такие же как в C/C++: && || !
  - Логическое И, ИЛИ и отрицание
- Эти операторы работают только со значениями типа boolean и возвращают значение типа boolean
- При вычислении логических выражений используется принцип укороченного вычисления
  - Например, в выражении *name != null && name.equals(“Donnie")* часть *name.equals("Donnie")* вычисляется только в случае, если *name != null == true*
- Приоритет логических операторов начиная с наивысшего:

  ! && ||

Строковые операторы

- Для конкатенации строк используется оператор +

```
public static void main(String[] args) {
String name = "Donnie";
System.out.println("Hello " + name);
}
```

- Конкатенация работает в случае если хотя бы один из двух ее операторов это строка.
  - Оператор + вычисляется слева направо:

```
int i = 5;
int j = 4;
System.out.println("i = " + i); // Печатает "i = 5"
System.out.println(i + j);   // Печатает "9"
System.out.println("i + j = " + i + j); // "i + j = 54"
System.out.println(i + j + " = i + j"); // "9 = i + j"
```

Управление исполнением кода в Java

- Операторы управления исполнением почти идентичны C/C++

```
if (условие)                        while (условие)
    выражение;                          выражение;
else if (условие)
    выражение;                      do
else                                    выражение;
    выражение;                      while (условие);
```

- Разница в том, что (условие) всегда возвращает значение типа boolean
- Блоки выражений так ж как в C/C++ заключаются в фигурные скобки

```
if (условие) {
    выражение1;
    выражение2;
}
```

Циклы for в Java

- Эти циклы также очень похожи на циклы C/C++
  - В цикле инициализируется и возможно сначала объявляется одна или несколько переменных цикла
  - Проверяется условие окончания цикла
  - Изменяются значения переменных цикла

```
for (инициализация; условие; изменение) выражение;
for (инициализация; условие; изменение) {
    выражение1;
    ...
}
```

- Более компактный эквивалент цикла while:

```
int i = 1;
while (i <= 10) {
    sum += i;
    i++;
}
```

```
for (i = 1; i <= 10; i++)
    sum += i;
```

Еще о циклах Java

- В цикле можно указать несколько начальных значений:

```
int i, sum;
for (i = 1, sum = 0; i <= 10; i++)
    sum += i;
```

- В цикле можно объявлять новые переменные цикла:

```
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
```

  - в этом примере переменная i существует только внутри цикла

И еще о циклах Java

- В цикле может быть несколько операторов изменения переменных:

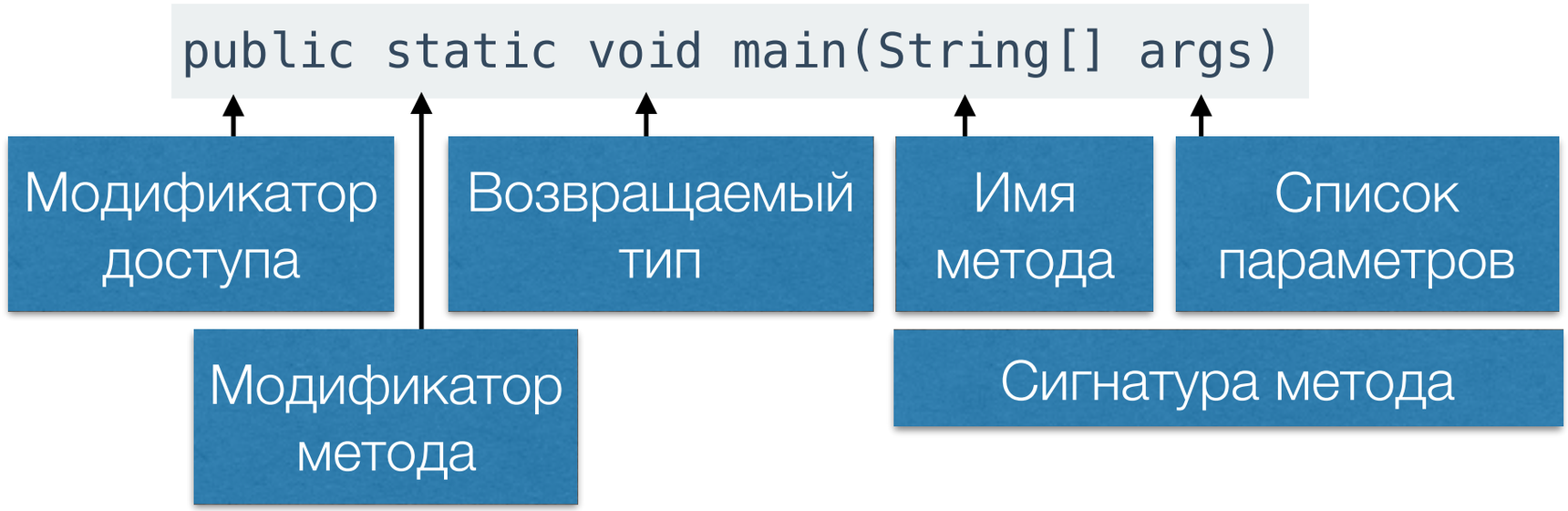
```
int sum = 0;
for (int i = 1; i <= 10; sum += i, i++) /* ничего */;
```

  - Обратите внимание, что в этом примере циклу не требуется тело.
- Ту же задачу можно решить с помощью более компактного кода:

```
int sum = 0;
for (int i = 1; i <= 10; sum += i++) /* ничего */ ;
```

  - Но, такой код плохо читается и, в целом, следует избегать подобные приемы написания программ

Методы Java - краткий обзор



- Метод возвращает значение указанного типа
- Или если в качестве типа указано void не возвращает ничего
- Метод может иметь любое число аргументов
  - Пустые скобки () указывают на отсутствие аргументов.
  - Сигнатура метода включает его имя и список параметров.
  - Модификаторы рассмотрим чуть позже.

Печать в Java

- Основная функция, использующаяся для печати System.out.println(“Hello!")
- Имеет много вариаций:

```
System.out.println(String x)
System.out.println(boolean x)
System.out.println(char x)
System.out.println(float x)
System.out.println(int x)
System.out.println(Object x)
System.out.println()
```

  - есть и другие варианты …
- Это пример переопределения метода
  - Тоже имя, но другая сигнатура

Ввод/вывод на консоль

- System.out – это стандартный выходной поток консоли
- System.err – стандартный поток ошибок
  - Используется для вывода сообщений об ошибках.
- System.in – стандартный входной поток
  - Воспользуемся им в следующей лекции.
- System.out.println(…) – печатает и переходит на следующую строку
- System.out.print(…) – оставляет курсор на текущей строке

Замечание об именах классов

- Классы Java могут быть сгруппированы в пакеты
  - Это делать не обязательно, но очень полезно
  - Пакеты формируют иерархию классов
- package1.package2.ClassName
  - Имена пакетов обычно указываются в нижнем регистре
  - Правила назначения имен те же, что для переменных
  - Пример: java.awt.event.MouseEvent
- Подробности позже!

Терминология: классы и объекты

- Java это объектно-ориентированный (ООП) язык
  - Программы полностью состоят из классов
- Объект это тесная комбинация двух вещей:
  - Состояние - набор логически связанных данных
  - Поведение - кода, который обрабатывает эти данные
- Класс это “шаблон” объектов
  - Класс определяет состояние и поведение объектов
  - Каждый класс это отдельный новый тип языка

Поля и методы класса

- Класс состоит из членов класса:
- Поля – переменные класса, хранящие состояние
- Методы – операции которые класс может выполнять
    - Набор методов класса определяет его поведение
    - Код метода называется реализацией метода
    - Методы обычно, хотя необязательно, используют поля класса

Особенные методы

- Конструкторы – используются для инициализации новых экземпляров класса
  - Конструктор может иметь аргументы, но не возвращает значения
  - Класс должен иметь по крайней мере один конструктор
- Аксессоры – используются для доступа к внутренним данным класса
  - Контролируют внешнее представление этих данных
- Мутаторы – изменяют внутренние данные класса
  - Контролируют когда и как могут быть изменены данные класса
- В Java нет деструкторов!
- Не все классы имеют аксессоры и мутаторы

Абстракция и инкапсуляция

- Абстракция
  - Класс предоставляет простой и понятный интерфейс
  - При этом не интересные детали реализации скрыты внутри класса
    - Пользователям класса не требуется разбираться в этих деталях
    - Пользователи могут сконцентрироваться на решении собственных задач с помощью класса
- Инкапсуляция
  - Позволяет защитить внутреннее состояние объекта от доступа и модификации извне
  - Объект контролирует изменение своего состояния
    - С помощью методов, которые могут обеспечить необходимые проверки, прежде чем выполнить это изменение

Модификаторы доступа

- Модификаторы доступа можно применять к классам, методам и полям
- В языке Java имеется четыре модификатора доступа:
  - public – каждый имеет право доступа
  - private – только сам класс имеет право доступа
  - protected – разберемся позже :)
  - Модификатор по умолчанию, то есть если модификатор явно не указан
    - Называется внутри-пакетным.
- Обязательно используйте модификаторы доступа, для того чтобы скрыть детали реализации вашего кода.

Пример

```
public class Point2d {
    // Координаты
    private double xCoord;
    private double yCoord;
    /** Конструктор с 2-мя аргументами. **/
    public Point2d(double x, double y) {
        xCoord = x;
        yCoord = y;
    }
    /** Конструктор по умолчанию; инициализирует нулями (0, 0). **/
    public Point2d() {
        // вызываем конструктор с 2-мя аргументами
        this(0, 0);
    }
    public double getX() { return xCoord; } // аксессор
    public double getY() { return yCoord; }
    public void setX(double x) { xCoord = x; } // мутатор
    public void setY(double y) { yCoord = y; }
}
```

Правила назначения имен методов

- Аксессоры Java обычно начинают со слова get:
  - double getX()
  - double getY()
- Мутаторы начинают со слова set:
  - void setX(double)
  - void setY(double)
- Аксессоры возвращающие значения типа Boolean часто начинают с шы:
  - boolean isRunning()
  - boolean isLoaded()
- Допускаются исключения, если is не имеет смысла:
  - boolean contains(Object)
  - boolean intersects(Set)

Пример работы с классом Point

- Создаем новый объект Point2d с помощью оператора new

```
Point2d p1 = new Point2d();
Point2d p2 = new Point2d(3.04, -5.612);
```

- Вызываем методы объекта

```
p1.setX(15.1);
p1.setY(12.67);
System.out.println("p2 = (" + p2.getX() +
    "," + p2.getY() + ")");
```

Объекты и ссылки

- Что такое p1 и p2?

```
Point2d p1 = new Point2d();
Point2d p2 = new Point2d(3.04, -5.612);
```

  - Это ссылки на объекты типа Point2d
    - Они не не являются сами по себе объектами
- Продолжим предыдущий пример:

```
Point2d p3 = p1; // по прежнему есть только два объекта
p1 = null; // оба объекта по прежнему доступны
p2 = null; // один из объектов больше недоступен!
```

- Виртуальная машина Java следит за количеством ссылок на объект. Если ссылок не остается, объект удаляется из памяти
  - Эта технология называется сборка мусора (Garbage collection)

Передача объекта через аргументы метода

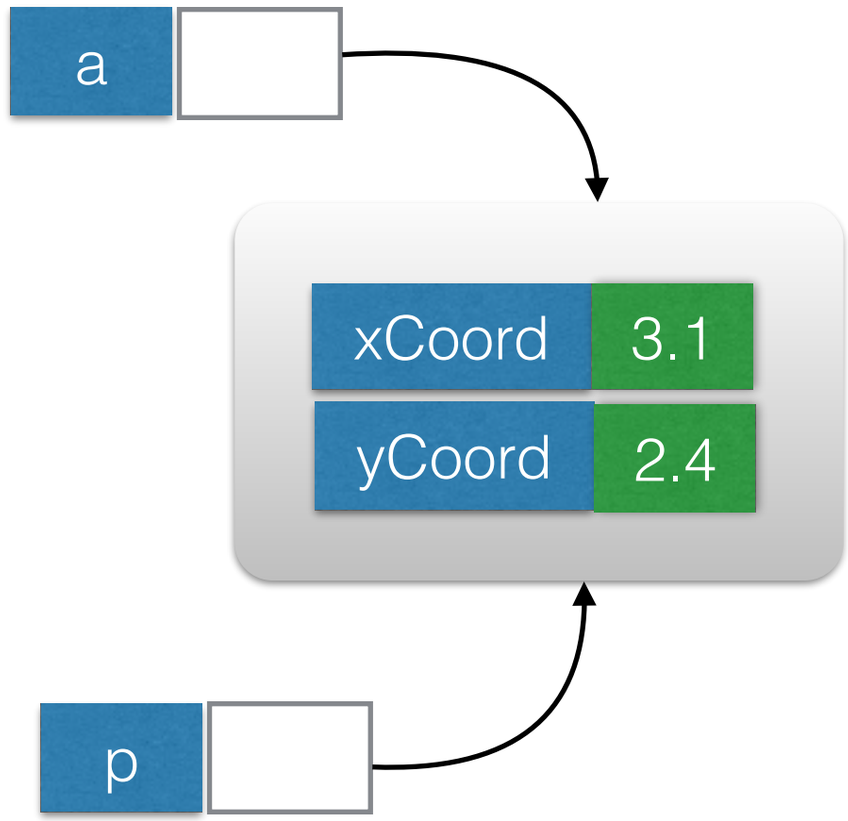
- Что происходит, если вы вызываете функцию с объектом в аргументах?

```
public void printPoint(Point2d p)
```

- Вспомним, что p это ссылка на объект
- В переменную копируется ссылка p, но не сам объект
- Возможны побочные эффекты и неожиданные баги

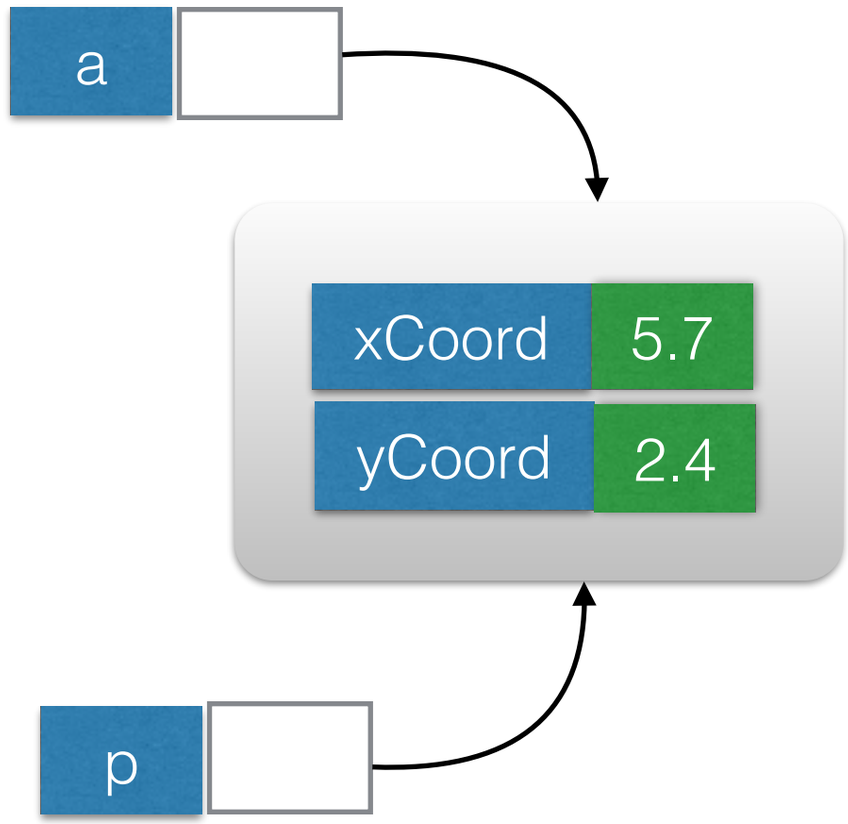
Передача объектов в Java

```
void main(String[] args) {
    Point2d a = new Point2d(3.1, 2.4);
    printPoint(a);
}

void printPoint(Point2d p) {
    System.out.println(p.getX() + "," +
        p.getY());
    p.setX(5.7); // ??
}
```



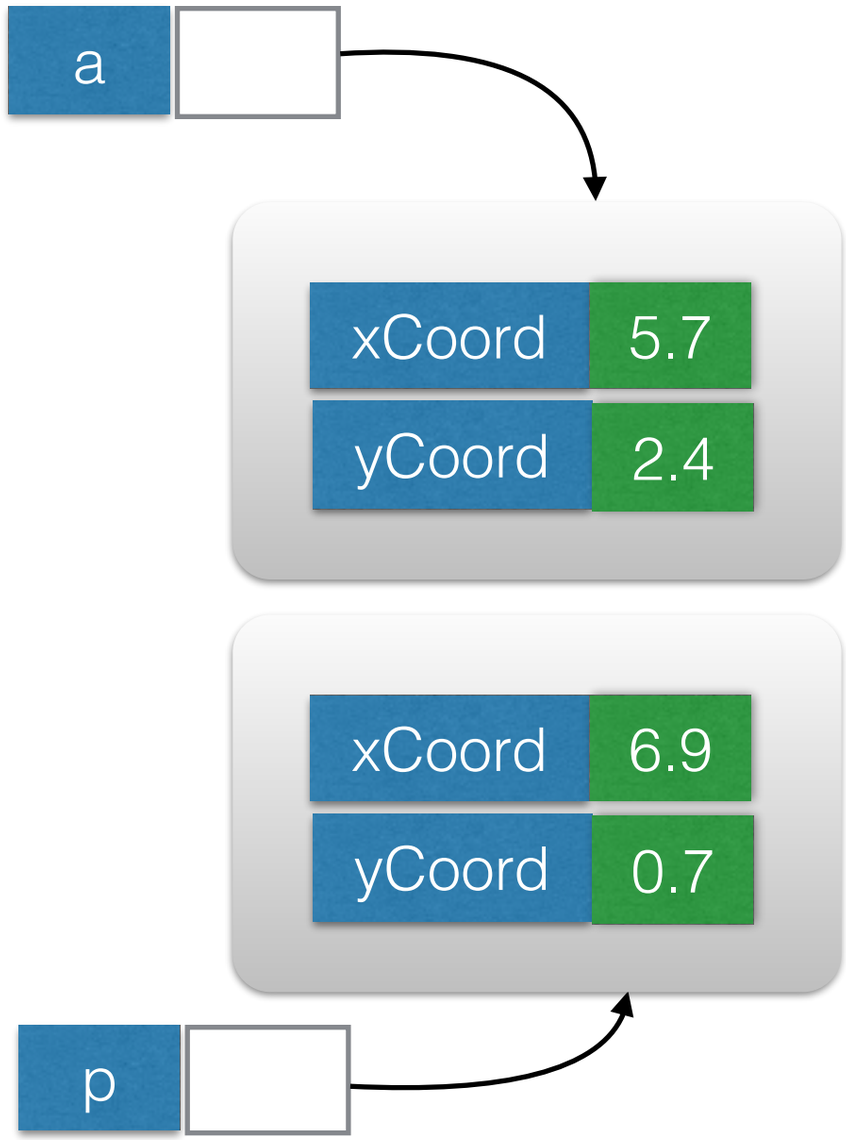
Передача объектов в Java (2)

```
void main(String[] args) {
    Point2d a = new Point2d(3.1, 2.4);
    printPoint(a);
}

void printPoint(Point2d p) {
    System.out.println(p.getX() + "," +
        p.getY());
    p.setX(5.7); // изменяет a
    p = new Point2d(-6.9, 0.7); // ???
}
```



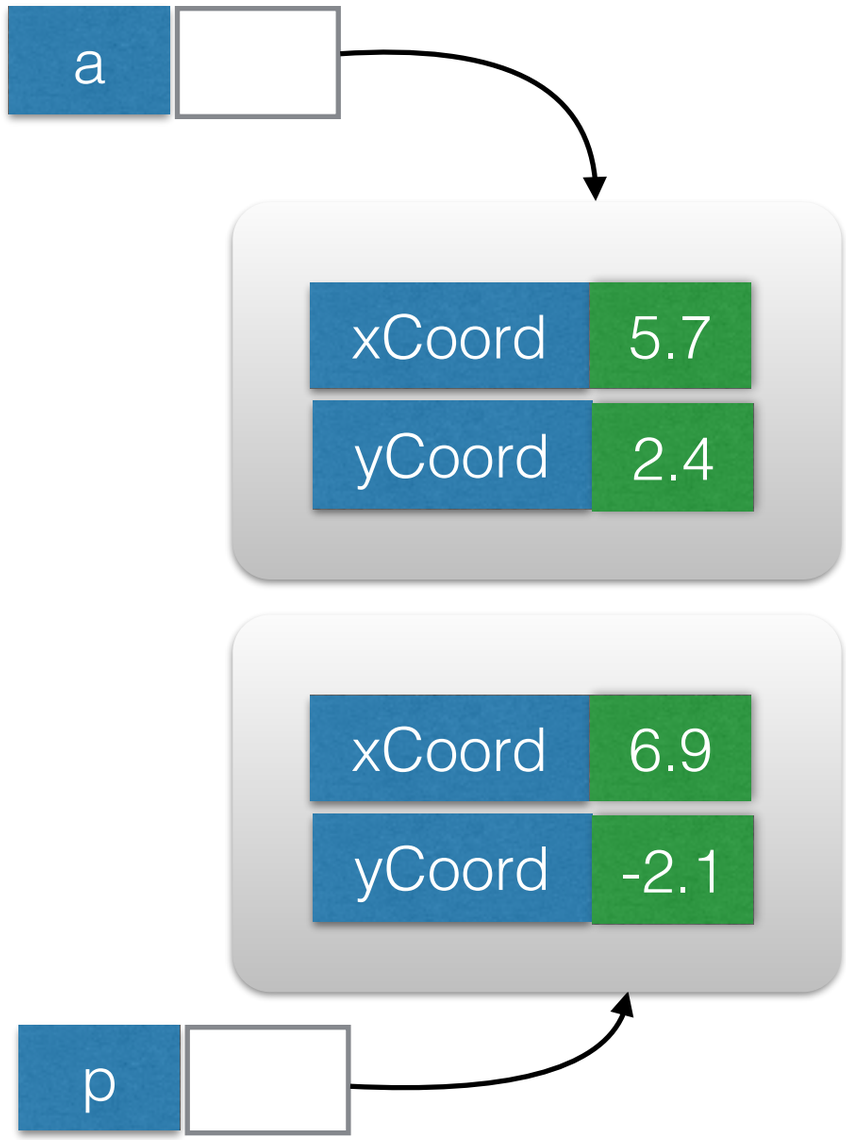
Передача объектов в Java (3)

```
void main(String[] args) {
    Point2d a = new Point2d(3.1, 2.4);
    printPoint(a);
}

void printPoint(Point2d p) {
    System.out.println(p.getX() + "," +
        p.getY());
    p.setX(5.7); // изменяет a
    p = new Point2d(-6.9, 0.7);
    p.setY(-2.1); // ???
}
```



Передача объектов в Java (4)

```
void main(String[] args) {
    Point2d a = new Point2d(3.1, 2.4);
    printPoint(a);
}

void printPoint(Point2d p) {
    System.out.println(p.getX() + "," +
        p.getY());
    p.setX(5.7); // изменяет a
    p = new Point2d(-6.9, 0.7);
    p.setY(-2.1); // меняет только
                  // локальную копию
}
```



Мораль

- Вы должны действовать очень аккуратно, когда имеете дело со ссылками на объекты
  - Если метод случайно по ошибке изменит объект, это будет довольно трудно обнаружить во время отладки
- Там где это имеет смысл, делайте объекты неизменяемыми
  - В Java нет эквивалента ключевому слову const C++
  - Объект нельзя изменить (точнее нельзя изменить состояние объекта) если у него нет мутаторов
    - Состояние объекта инициализируется во время его создания
    - Позаботьтесь о том, чтобы не было других способов изменить его состояние

this

- Большинство методов имеет неявный (то есть явно не указанный) параметр this
  - this это ссылка на сам вызываемый объект
- Этот параметр неявно используется при обращении к полям объекта в методах

```
public double getX() {
    return xCoord; // тоже что "return this.xCoord;"
}
public String toString() {
    // тоже что "this.getX()" и "this.getY()"
    return "(" + getX() + " " + getY() + ")";
}
```

this (2)

- this следует использовать в случае необходимости разрешить конфликт имен

```
void setX(double xCoord) {
    // xCoord это параметр
    // this.xCoord это поле объекта
    this.xCoord = xCoord;
    }
```

- Такое часто встречается в реализации мутаторов
  - Имя аргумента совпадает с именем поля
- В целом же, следует избегать конфликтов имен
  - Все это может порождать трудно выявляемые ошибки

Статические методы

- Некоторые методы могут вызываться без привязки к объекту
  - Они называются статическими методами или методами класса

```
public static double atan2(double y, double x);
```

  - Статические методы не могут использовать ссылку this
    - Метод не вызывается для какого то определенного объекта
  - Для вызова статического метода следует указать имя его класса ClassName.methodName()
- Не статические методы называются методами экземпляра класса
- Класс java.lang.Math имеет только статические методы:

```
double tangent = Math.atan2(yComp, xComp);
```

Равенство в языке Java

- Примитивные типы используют оператор == так как можно ожидать
- Но для ссылочных типов оператор == сравнивает указатели

```
Point2d p1 = new Point2d(3, 5);
Point2d p2 = new Point2d(3, 5);
Point2d p3 = p1;
```

- Здесь p1 и p3 ссылаются на один и тот же объект, а значит:
  - p1 == p3 равно true, и
  - p1 == p2 равно false, даже если значения объектов одинаковые (но объекты разные, поэтому false).
- Для сравнения значений объектов используется метод obj.equals(obj2)
  - Создавая новый класс, можете сами написать подходящий для него метод equals()

Метод equals()

- Сигнатура

```
public boolean equals(Object obj)
```

- Метод возвращает true если obj “равен” объекту this
  - Если obj равен null всегда возвращается false
- Заметим что obj это объект типа Object
  - это универсальная ссылка на объект любого типа
  - Тип объекта можно получить с помощью оператора instanceof.

Свойства equals()

- Рефлексивность
  - a.equals(a) возвращает true
- Симметричность
  - a.equals(b) равно b.equals(a)
- Транзитивность
  - если a.equals(b) равно true и b.equals(c) равно true, то a.equals(c) тоже равно true
- Сравнимость с null
  - a.equals(null) возвращает false

Равны ли эти объекты Point?

```
@Override
public boolean equals(Object obj) {
    // obj это Point2d?
    if (obj instanceof Point2d) {
        // приведем объект other к типу Point2d,
        // и затем сравним.
        Point2d other = (Point2d) obj;
        if (xCoord == other.getX() &&
                yCoord == other.getY()) {
            return true;
        }
    }
    // Если мы попали сюда, объекты не равны.
    return false;
}
```

Оператор instanceof

- Используется для определения типа (класса) объекта
- Возвращает false, если ссылка равна null
  - Поэтому в нашем примере не надо отдельно проверять аргумент функции на равенство null

Почему equals(Object)?

- Классы могут наследоваться от других классов
  - Классы потомки наследуют все поля и методы родительского класса
  - Так можно создавать иерархические библиотеки классов
  - С объектом класса потомка можно работать так как будто это объект его родительского класса, по тому что он (как минимум) имеет все поля и методы этого родительского класса
- В языке Java все классы наследуются от класса java.lang.Object.
  - Следовательно, со всеми объектами любого класса можно работать так как будто они имеют тип Object
  - java.lang.Object реализует функционал, который предоставляется всем другим классам Java
    - Например методы equals(), hashCode(), getClass(), clone() и другие
  - Метод equals() можно использовать для сравнения объектов любого типа

Документация Java API

- Это полная документация API всей платформы Java
  - Очень полезна, особенно после того как вы научитесь с ней работать
  - Автоматически извлекается из исходного кода библиотек Java
- Содержит информацию о всех классах и интерфейсах
  - Правила их использования
  - Набор предоставляемых свойств
  - Связи между ними
- http://java.sun.com/javase/6/docs/api
  - Возможно, вам лучше сделать копию у себя на компьютере

Другая полезная документация

- Учебник Java
  - Содержит примеры проектирования программ охватывающие различные темы
  - Очень полезен для изучения новых возможностей
- Документация Java Development Kit (JDK)
  - Изменения и новые возможности
  - Документация инструментов проектирования
- Спецификация языка Java
- Спецификация виртуальной машины Java

Задание

- Создайте свою первую программу на языке Java
  - Формула Герона
  - Создайте класс, описывающий трехмерную точку, и добавьте методы distanceTo() и equals()
  - Создайте другой класс, который по трем заданным точкам вычисляет площадь треугольника по формуле Герона
- Изучите процесс компиляции и запуска программ на языке Java